Tech Info(vol5)

디스크 상의 DECIMAL 데이터 형식

디스크에 DECIMAL(값, 지수, 부호) 형식이 저장되는 방법과 필요한 공간에 대하여 알아보고자 합니다.

대부분의 필요한 정보는 "decimal.h"에 있으나 매우 주의 깊게 읽어야만 이해할 수 있을 것입니다.

이 글에서 표현되는 표기법은 다음과 같습니다.

0xAB의 형식은 16진수를 나타내며,
%1101의 형식은 이진수입니다.
기타 다른 형식은 모두 십진수로 표현된 것입니다.
1.23E45는 1.23*(10의 45제곱)입니다.

DECIMAL과 MONEY는 저장 형식이 동일합니다. 사실상 DATETIME과 INTERVAL도 DECIMAL 값으로 저장되나 여기서 다루지는 않겠습니다.

DECIMAL(m) 값의 경우, 실제 디스크 공간은 N = (m + 1) / 2 + 1 바이트가 필요하고, DECIMAL(m, n) 값의 경우에는 N = (m - n + 1)/2 + (n + 1)/2 + 1 바이트가 필요합니다.

나눈 결과는 몫만 취하고 나머지는 무시합니다. 그러나 여기서 중요한 것은 DECIMAL(m)과 DECIMAL(m, 0)을 저장하는 데 필요한 공간은 같으나, DECIMAL(m)과 DECIMAL(m, 0)이 나타낼 수 있는 값의 범위는 크게 다르다는 것입니다.

예를 들어, DECIMAL(2)와 DECIMAL(2,0)의 자료형을 갖는 두 칼럼이 있다고 가정한다면, 첫 번째 칼럼에는 두 자리 이상의 값도 INSERT될 수 있으나 두 번째 칼럼의 경우에는(예를 들어 123.0을 입력하면) 오류(1226: Decimal or money value exceeds maximum precision)가 발생합니다.

DECIMAL 형의 자료가 저장되는 N의 첫째 바이트는 1개의 부호 비트(양수는 1, 음수는 0)와 7개의 지수 비트(실제로는 65 초과 표기법으로 저장)의 두 부분으로 구성됩니다. 숫자의 부호가 음수이면, 나머지 7개의 지수 비트에서는 0과 1을 뒤바꿉니다. 즉 지수의 보수를 취하는 것입니다.

예를 들어, DECIMAL 값이 6.4E14일 경우 밑이 100인 지수로 표시하면 06.40 * (100 ** 7)로 표시되지만, 지수 7은 여기에 65를 더한 값으로 저장되므로 72, 즉 0x48(%01001000)이 됩니다. 이 때 양수는 부호 비트인 첫 번째 비트가 1이므로 첫 번째 비트를 1로 하여 이 DECIMAL 값을 위한 바이트는 %11001000, 즉 0xC8입니다.

만일 DECIMAL 값이 -6.4E14와 같이 음수일 때는 부호 비트가 0이고 지수의 비트를 뒤바꾸어 저장하므로, 첫째 바이트의 값은 %00110111, 즉 0x37입니다.

또한, 지수가 음수(+6.4E-14)이면 지수 비트의 값은 65 - 7 = 58 = 0x3A이고 첫 번째 바이트는 %10111010, 즉 0xBA가 되며, 숫자와 지수가 모두 음수(-6.4E-14)이면 첫째 바이트의 값은 %01000101, 즉 0x45가 됩니다.

다시 정리하여 살펴본다면 위에서 예로 들어 설명한 DECIMAL 값은 다음과 같은 부호와 지수 바이트를 갖습니다.

+6.4E+14: 0xC8
-6.4E-14: 0x45
+6.4E-14: 0xBA
-6.4E+14: 0x37

이렇게 살펴볼 때 DECIMAL(2) 형식으로 저장된 1은 0xC1 0x01 바이트로 표시될 것입니다.

값의 나머지 바이트는 주어진 DECIMAL 값의 밑이 100인 유효 숫자로 이루어지며, DECIMAL(4) 칼럼은 (4+1) / 2 + 1 = 3바이트가 필요하며, 이 칼럼에 6.4 * 10 ** 14 값을 INSERT하게 되면 실제로 디스크에 0xC7, 0x06, 0x28으로 저장됩니다. 즉 이 값을 다시 쓰면 06.40*100**7로 표시할 수 있으므로, 두 번째 바이트는 06이라는 정수 부분을 표시한 것이고, 세 번째 바이트는 소수 부분, 즉 40(16진수로는 0x28)을 나타냅니다.

숫자가 음수이면 유효 숫자는 100의 보수 표기법으로 저장됩니다.

100의 보수 연산은 두 가지 경우 중 하나입니다. 이 경우 음수는 99(0x63)에서 유효 숫자를 뺀 밑이 100인 숫자로 저장되나, 최하위 숫자는 여기에 0x01을 더한 값이 되고, 이 때 올림이 있으면 유효 숫자 자리가 늘어납니다.

따라서 -6.4E+14는 0x37, 0x5D(99-6 = 십진수 93), 0x3C(99-40+1 = 60, 최하위 숫자이므로 1을 더함)로 저장됩니다.

DECIMAL(4)에서 0은 0x80, 0x00, 0x00으로 표시됩니다.

DECIMAL(4)에서 Null은 0x00, 0x00, 0x00으로 표시됩니다.

ECIMAL(4)에서 1은 0xC1 0x01 0x00으로 표시됩니다.

DECIMAL(4)에서 -1은 0x3E 0x63 0x00으로 표시됩니다.

(-1은 -1*100**0으로 표시할 수 있으며, 우선 지수 비트 7자리는 %1000001이고 이 값을 보수로 취하면서 부호 비트인 첫 번째 비트를 음수인 0으로 채우게 되므로 %00111110, 즉 0x3E가 됩니다. 그리고 두 번째 바이트는 음수를 저장하는 방식에 따라 99-1+1=99, 즉 0x63이 저장되게 됩니다.)

이렇게 알아보기 힘든 표시 형식을 사용하는 이유는 이 표기법으로 두 DECIMAL을 비교하는 것이 훨씬 간단하기 때문입니다. 같은 칼럼의 두 수를 부호 없는 문자형으로 비교하면, 단지 바이트 순서에 따라 정렬할 수 있습니다.

다음은 위의 정보에 따라 DECIMAL(4)의 해당 십진수 값을 나타낸 것입니다.

첫째 바이트 처음 1개 비트 ; 부호 비트

0=음수, 1=양수

나머지 7개 비트 = 65 초과 형식 지수

나머지 바이트 100의 보수 형식의 밑이 100인 유효 숫자

========== -06.41E+14: 0x37 0x5D 0x3B -06.40E+14: 0x37 0x5D 0x3C -06.41E-14: 0x45 0x5D 0x3B -06.40E-14: 0x45 0x5D 0x3C +06.40E-14: 0xBA 0x06 0x28 +06.41E-14: 0xBA 0x06 0x29 +06.40E+14: 0xC8 0x06 0x28 +06.41E+14: 0xC8 0x06 0x29 00.00 : 0x80 0x00 0x00 NULL : 0x00 0x00 0x00 1 : 0xC1 0x01 0x00 -1 : 0x3E 0x63 0x00 ==========

DataBlade API를 이용한 사용자 정의 함수

DataBlade API(Application Programming Interface)를 이용하여 사용자 정의 함수(User-defined Function)를 생성하는 방법을 알아보도록 하겠습니다.

DataBlade API:

DataBlade API는 Server API라고도 이야기합니다. DataBlade API는 Informix Dynamic Server Universal Data Option 데이터베이스 내에 저장되어 있는 데이터에 접근하는 서버 함수를 개발하는데 사용할 수 있습니다. 지금까지는 SPL을 이용하여 서버 함수를 많이 작성하여 사용해 왔지만 SPL이 주는 여러 가지 제약 사항, 즉 데이터베이스 서버 외부에 있는 루틴의 실행 불가, 새로운 기반 타입(OPAQUE TYPE)의 생성 불가, 대형 객체를 다루는 루틴의 생성 불가, 동적인 SQL구문의 생성 불가 등과 같은 제약 사항이 있었으나 C, C++ 등으로 외부 함수를 작성하여 제약 없이 사용할 수 있다는 커다란 장점이 있습니다.

외부 함수:

사용되는 함수가 내장 프로시저(SPL)가 아닌 C, C++ 혹은 Java로 작성된 경우 외부 함수라 하는데, 사용자 정의 함수는 사용자 정의 함수에 대한 연산을 수행하고 응용 로직을 캡슐화하는 데 사용합니다.

외부 함수는 함수가 될 수도 있고 프로시저가 될 수도 있습니다. 함수건 프로시저건 서버 상에 공유 라이브러리(shared library) 형태로 저장됩니다. 그리고 IUS가 해당 함수를 실행하면 IUS는 공유 라이브러리에 있는 오브젝트

코드를 호출하게 됩니다.

외부 함수의 등록:

외부 함수를 컴파일하여 공유 라이브러리로 만들면 이는 반드시 IUS에 등록을 해야만 사용할 수 있습니다. 외부 함수를 등록하는 방법은 CREATE FUNCTION이나 SPL의 CREATE PROCEDURE 구문과 아주 흡사한 구문을 이용하여 등록을 행합니다. 다음은 외부 함수를 등록하는 구문의 문법입니다.

```
CREATE FUNCTION routine_name(parameter_list)

        RETURNING type_name;

        EXTERNAL NAME 'path_name'

        LANGUAGE C

END FUNCTION;
```

외부 함수 수정자(Modifiers):

외부 함수 작성 시 선택적으로 추가할 수 있는 여러 가지 수정자가 있는데, 다음은 이와 관련한 구문의 문법을 보인 것입니다.

```
CREATE FUNCTION routine_name(parameter_list)

        RETURNING type_name

        WITH    (NOT VARIANT,

                HANDLESNULL,
```

```
                CLASS='myclass',

                INTERNAL,

                STACK=stacksize);

        EXTERNAL NAME 'path_name'

        LANGUAGE C

END FUNCTION;
```

HANDLESNULLS:

이 수정자는 함수의 입력 매개변수가 NULL인 경우 실행하고 적절한 처리를 요하는 경우 사용하는 수정자입니다. HANDLESNULLS를 지정하지 않고 NULL 값을 입력 매개변수로 전달하면 함수는 NULL을 리턴합니다. 기본적으로 HANDLESNULLS는 외부 함수에는 설정되지 않습니다. 즉, IUS는 외부 함수가 NULL 인수를 처리하지 않음을 의미합니다.

VARIANT / NOT VARIANT:

이 수정자는 함수가 동일한 입력 매개변수를 가지고 호출되었을 때 다른 결과값을 산출할 수도 있음을 의미합니다. (즉, 처리 결과가 함수를 호출한 시간에 따라 다르게 나타나는 경우를 말합니다.) 이 정보는 옵티마이저가 외부 함수를 참조할 때마다 함수를 실행해야 할 것인가를 결정하기 위해 사용합니다. VARIANT 수정자는 함수를 작성하는데 사용한 언어의 절 (clause)이나 수정자 목록이 될 수 있습니다. 이 수정자는 프로시저에는 사용할 수 없습니다. 기본적으로 사용자 정의 함수는 VARIANT 수정자를 사용합니다. Non-VARIANT 함수를 정의하려면, CREATE FUNCTION 문의 WITH 절에 NOT VARIANT를 지정하면 됩니다.

CLASS:

이 수정자는 사용자가 정의한 가상-프로세서 클래스에서 함수를 실행하도록 합니다. 기본적으로 사용자 정의 함수는 CPU VP 클래스에서 실행합니다.

STACK:

이 수정자는 IUS가 STACKSIZE 환경 구성 매개변수에서 설정한 스택의 크기보다 큰 스택에서 함수를 실행하게 합니다. STACK을 사용하면 스택의 크기는 함수가 실행되는 동안 지정한 바이트 수만큼 증가하게 되며, 함수의 실행이 종료되면 본래의 크기에 해당하는 스택을 반환합니다.

INTERNAL:

이 수정자는 SQL이나 SPL 구문에서 함수를 호출하지 못함을 지정하는 것으로, INTERNAL을 지정하지 않으면 SQL이나 SPL 구문에서 기본적으로 함수를 호출할 수 있습니다.

1. 외부 함수 작성

먼저 해야 할 일은 C 코드를 작성하여 이를 공유 라이브러리 형태로 컴파일하는 것입니다. 아래는 C 코드로 작성한 함수의 본문 예를 보인 것인데, 함수를 작성하는 경우에는 반드시 IDS/UDO DataBlade API Programmer's Manual이나 DataBlade Developer Kit User's Guide를 참조해야 합니다.

```
/*

** Title:  c581.c

** SCCSid:          %W% %E% %U%

** CCid: %W% %E% %U%
```

```
** Author:

** Created:          Jun 23, 1998 17:21:30 PM

** Description:      Generated 'C' file.

** Comments:         Generated for project c581.1.0

*/




/*

** The following is placed here to insure

** that name "mangling" does not occur.

*/




#ifdef __cplusplus




extern "C"
```

```
{




#endif



/* Standard library includes. */

#include

#include

#include

#include



/* Used by Informix GLS routines. */

#include



/* Inlcude when accessing the Informix API. */
```

```
#include



/* This is the project include file. */

#include "c581.h"



/*************************************************************

**

** Function name: volumn_c

** Description:

** Special Comments:

**          Entrypoint for the SQL routine volumn_c(dimensions).

** Parameters:

** Return value:     mi_double_precision *

** History:
```

```
**          Jun 23, 1998 ? Generated by BladeSmith Version 3.20.TGN90.

** Identification:

** Warning: Do not remove or modify this comment:

**          volumn_c

** FunctionID: 82030652-b5c9-11d0-b2ed-080009d3c6e5*/

************************************************************

*/

mi_double_precision *

volumn_c

(

MI_ROW *parm1,

MI_FPARAM *Gen_fparam                   /* Standard info ? see DBDK docs. */

)

{
```

```
MI_CONNECTION *Gen_Con;          /* The connection handle. */

mi_double_precision *length_c;

mi_double_precision *width_c;

mi_double_precision *height_c;

mi_double_precision *Gen_RetVal;  / The return value. */

mi_integerlengthlen = 0, widthlen = 0, heightlen = 0;



Gen_RetVal = mi_alloc(sizeof(mi_double_precision));



/* Get the current connection handle. */

Gen_Con = mi_open(NULL, NULL, NULL);



/* Verify that the connection has been established. */

if (Gen_Con == 0)
```

```
{

        /*

        ** Opening the current connection has failed

        ** so issue the following message and quit.

        **

        ** "Connection has failed in volumn_c."

        */

        mi_db_error_raise(Gen_Con, MI_SQL, ERRORMESG1,

            "FUNCTION%s", "volumn_c", (char *)NULL);




                /* not reached */

}

mi_value_by_name(parm1, "length", (MI_DATUM *)&length_c, &lengthlen);

mi_value_by_name(parm1, "width", (MI_DATUM *)&width_c, &widthlen);
```

```
    mi_value_by_name(parm1, "height", (MI_DATUM *)&height_c, &heightlen);




    if (lengthlen == 0 || widhtlen == 0 || heightlen == 0)

    {

            *Gen_RetVal = 0;

    }

    else

    {

            *Gen_RetVal = (*length_c) * (*width_c) * (*height_c);

    }

    return Gen_RetVal;

}

/* Warning: Do not modify. Volumn_c checksum: 937557566 */

#ifdef __cplusplus
```

```
}

#endif
```

2. 데이터베이스에 외부 함수 등록

C 코드로 함수를 작성한 후 반드시 데이터베이스에 작성한 함수를 등록해야 합니다. 위에서 작성한 volumn_c 함수를 등록하려면 아래와 같은 구문을 실행하면 됩니다.

```
CREATE FUNCTION volumn_c (parm1 dimensions)

                    RETURNING FLOAT;

                    EXTERNAL NAME

                      '$InformixDIR/extend/aaa.1.0/aaa.bld(volumn_c)'

                    LANGUAGE C

END FUNCTION;
```